# PULSE-WIDTH MODULATION (PWM) EXAMPLE CODE FOR A89211/12/24 FAMILY SOC

By M. Pramanathan
Allegro MicroSystems

## PULSE-WIDTH MODULATION (PWM) UNIT

The top-level architecture of the PWM module is shown in Figure 1. The clock frequency ($f_{PWM}$) of this module is 80 MHz. The register settings are set by the AMBA high-performance bus (AHB). The input control setting from the advanced motor control timer module (AMCT) can be used for Hall-sensor-based and BEMF-based control algorithms. The PWM outputs of this module are connected to a general-purpose input-output (GPIO) driver, which connects the PWM signals directly to the gate driver unit (GDU) in order to drive the external MOSFETs.

The analog-to-digital converter normal and calibration trigger signals (ADCANT and ADCACT) for each PWM phase are connected to the current acquisition unit (CAU) to sample the current at a desired point with respect to the generated PWM period cycle, making it suitable for field-oriented control (FOC) applications.

The PWM generator is capable of generating complementary PWM outputs with two modes of operation: center-aligned and edge-aligned. There is also a dedicated dead time control unit to ensure that no short circuit occurs when the complementary signals Hx and Lx are used to drive the external MOSFETs. There is also a PWM_Centre signal which is set high at the center of the PWM cycle and is set low at the start of the PWM cycle. This signal is connected to the digital acquisition unit (DAU), advanced motor control timer unit (AMCT), and GPIO.

The clock source ($f_{PWM}$) to the operating timer is 80 MHz. The timer $t_{PWM}$ contains a 14-bit counter that is used as a reference signal for the compare and event trigger control units. The timer has two modes of operation: center-aligned and edge-aligned. In both modes, the period of the timer $t_{PWM}$ is defined by the PERIOD parameter in the PWM PERIOD register.

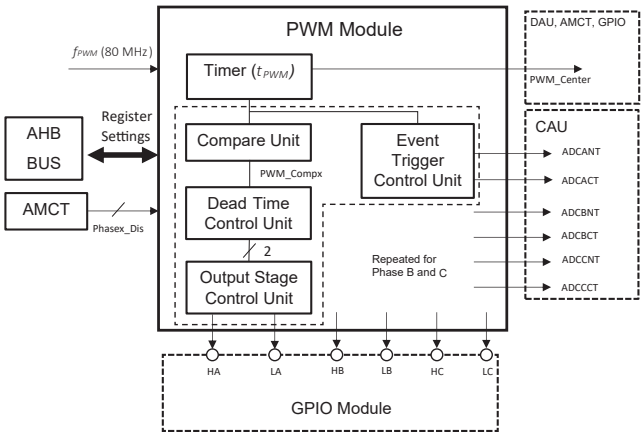


*Figure 1: Block Diagram of the PWM Module*

## PWM EXAMPLE CODE

Board: A89224EV EVM Rev0

Debugger: ULINK2 plus Cortex Debugger

The PWM example code demonstrates the following scenarios:

- Center-aligned duty cycle generation
- Edge-aligned duty cycle generation
- Force all low-sides on
- ADC normal trigger signal
- ADC calibration trigger signal

Further explanations of each scenario will be provided in the sections below. Details about the microcontroller abstraction layer (MCAL) functionality can be found under the Doxygen API Reference Guide available in the MCAL releases.

## Initialize PWM Output GPIO

The GPIO10 to GPIO15 are used for the gate driver logic signals die-to-die interface and must always be configured as output (PIN_MODE = 0101b). The gate driver logic level signals are generated by the PWM unit and placed in ESRC array. Therefore, ESRC array must be used for these pins (FOSEL = 1). ESRCSEL must be set to 0000b to ensure a correct connection of signals from the PWM unit to the gate driver.

```
void InitialisePwmGpio( void )
{
    // Set Up Gpio Pins for die to die Pwm Interface
    // Create GPIO 10-15 Objects for inter die to die PWM Connection
    Gpio_SetPinActive( eGPIO_INSTANCE_ID10 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID11 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID12 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID13 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID14 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID15 );

    // Set GPIOx Control Register to push-pull output.
    Gpio_SetMode( eGPIO_INSTANCE_ID10, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID11, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID12, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID13, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID14, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID15, eGPIO_MODE_OUTPUT_PUSH_PULL );

    // Set GPIO_USRCx to have FOSEL = 1 and ESRCSEL = 0
    Gpio_SetSource( eGPIO_INSTANCE_ID10, eGPIO_ESRC0 );
    Gpio_SetSource( eGPIO_INSTANCE_ID11, eGPIO_ESRC0 );
    Gpio_SetSource( eGPIO_INSTANCE_ID12, eGPIO_ESRC0 );
    Gpio_SetSource( eGPIO_INSTANCE_ID13, eGPIO_ESRC0 );
    Gpio_SetSource( eGPIO_INSTANCE_ID14, eGPIO_ESRC0 );
    Gpio_SetSource( eGPIO_INSTANCE_ID15, eGPIO_ESRC0 );
}
```

## Initialize GDU interface

The steps to initialize the GDU are as follows:

1. Initialize GDU master
2. Enable stop-on-fault for linear regulator, overtemperature, MOSFET overcurrent, and VREG
3. Configure control register
4. Disable multiplex with analog input source and mask the bootstrap undervoltage fault
5. Clear diagnostic and status register

```
void InitialiseGduInterface( void)
{
    uint16 u16TxData;
    const uint16 ku16CsoEn = 1 << 12;
    const uint16 ku16RegOvRride = 1 << 13;
    // 800mV Sense Amp Offset
    const uint16 ku16SenseAmpOffset = 3 << 5;
    // Gain 10
    const uint16 ku16SenseGain = 0 << 8;
    // 600mV Vds
    const uint16 ku16VdsThreshold = 1 << 1;
    // 0ns DeadTime
    const uint16 ku16Deadtime = 0 << 3;

    GduMaster_Initialize();

    // Enable Stop on Fault for Linear Regulator, over temperature,
MOSFET over current, and VREG
    u16TxData = 0x4D;
    GduMaster_Write( eSTOP_ON_FAULT, u16TxData );

    u16TxData = ku16RegOvRride | ku16CsoEn | ku16Deadtime |
ku16SenseAmpOffset | ku16SenseGain | ku16VdsThreshold;
    GduMaster_Write( eCONTROL, u16TxData );

    // Disable Multiplexer with analog input source
    u16TxData = 1 << 5;
    GduMaster_Write( eANALOG_MUX, u16TxData );

    // Mask Bootstrap undervoltage fault
    u16TxData = 1 << 2;
    GduMaster_Write( eMASK, u16TxData );

    // Clear Diagnostics
    u16TxData = 0xFFFF;
    GduMaster_Write( eDIAGNOSTIC, u16TxData );

    // Clear Status Register
    GduMaster_Write( eSTATUS, u16TxData );
}
```

## Boot Capacitor Charge

The A89211/12/24 product family does not include an internal charge pump, which is a hardware feature that automatically charges the bootstrap capacitor. Therefore, a manual pre-charge sequence must be implemented in software at startup. This ensures the high-side gate driver has enough voltage to properly turn on its MOSFET.

The code snippet to charge the boot capacitors is as follows:

```
#define BOOT_CAP_CHARGE 100
uint8 u8BootCapChargeCount = 0;
//bootcap charge
PwmGeneration( ePWM_CONTROL_REGISTER_ALIGNMENT_CENTRE_ALIGNED, PWM_
PERIOD_20KHZ, 0, 0, 0);
```

```
// wait until 1 PWM cycle is elapsed to charge bootcaps
while(u8BootCapChargeCount<BOOT_CAP_CHARGE)
{
    u8BootCapChargeCount++;
}
```

## Center-Aligned Duty Cycle Generation

This example configures the output PWM period to 20 kHz by assigning PERIOD to 4000 in the PWM period register, sets the PWM mode to center-aligned by configuring PWMMOD bit to 1 in control register 1 and configures the dead time to 500 ns by assigning DeadTime to 40 in PWM dead time register. It supports programmable duty cycles as shown in Table 1.

*Table 1: Programmable Duty Cycles*

| Phase | Duty Cycle | Register |
|---|---|---|
| A | 25% | PWM_DUTY_CYCLE_20KHZ_25 |
| B | 50% | PWM_DUTY_CYCLE_20KHZ_50 |
| C | 75% | PWM_DUTY_CYCLE_20KHZ_75 |

DutyCycle1x and DutyCyle2x are programmed by the PWM duty cycle register.

The duty cycle of the PWM_Compx signal is calculated as follows, depending on the Phasex_Inv signal.

When Phasex_Inv = 0:

$$\mathrm{PWM}_{\mathrm{Compx(Duty\ Cycle)}} = \left( \frac{\left( \frac{\mathrm{DutyCycle1x}}{2} + \frac{\mathrm{DutyCycle1x}}{2} \right)}{\mathrm{PERIOD}} \right) \times 100\,\%$$

When Phasex_Inv = 1:

$$\mathrm{PWM}_{\mathrm{Compx(Duty\ Cycle)}} = \left( \frac{\left( \frac{\mathrm{PERIOD-DutyCycle1x}}{2} + \frac{\mathrm{PERIOD-DutyCycle1x}}{2} \right)}{\mathrm{PERIOD}} \right) \times 100\,\%$$

For example, if it is desired to achieve 25% duty cycle for a $t_{PWM}$ period of 50 µs, DutyCycle1x and DutyCycle2x can be calculated as below when Phasex_Inv = 0.

The PERIOD should be 4000 to achieve TPWM period of 50 µs.

$$\mathrm{DutyCycle1x} + \mathrm{DutyCycle2x} = (25 \times 2 \times 4000) \,/\, 100 = 2000$$

In order to preserve symmetry, DutyCycle1x = DutyCycle2x.

Therefore:

$$\mathrm{DutyCycle1x} = \mathrm{DutyCycle2x} = 1000$$

Finally, the output is decided by independent PWM generators when OutputSelection register is set to value 0x3F.

### Important Notes:

- If PERIOD, DutyCycle1x, or DutyCycle2x are odd numbers, they are rounded down to the closest even number.
- If DutyCycle1x or DutyCycle2x are greater than PERIOD, they are respectively clamped to PERIOD.
- The operations start once the PERIOD set to a value greater than 15d.

To execute this configuration, ensure that the variable u8Test is set to PWM_CENTRE_ALIGNED_DUTY_CYCLE_GENERATION.

```
Std_ReturnType PwmGenExample_Run()
{
    uint8 u8Test;
    uint32 u32CentreAlignPhADutyCycle;
    uint32 u32CentreAlignPhBDutyCycle;
    uint32 u32CentreAlignPhCDutyCycle;
    // Test type
    u8Test = PWM_CENTRE_ALIGNED_DUTY_CYCLE_GENERATION;
    // Disable watchdog
    Scu_WatchdogDisable();
    // Initialise PWM GPIO
    InitialisePwmGpio ();
    // Initialise GDU
    InitialiseGduInterface();

    switch ( u8Test  )
    {
        case PWM_CENTRE_ALIGNED_DUTY_CYCLE_GENERATION:
         u32CentreAlignPhADutyCycle = (uint32)( PWM_DUTY_CYCLE_20KHZ_25
+ ((uint32)((uint32)PWM_DUTY_CYCLE_20KHZ_25 << (uint8)ePWM_DUTY_CYCLE_
MASK2_SHIFT )));
         u32CentreAlignPhBDutyCycle = (uint32)( PWM_DUTY_CYCLE_20KHZ_50
+ ((uint32)((uint32)PWM_DUTY_CYCLE_20KHZ_50 << (uint8)ePWM_DUTY_CYCLE_
MASK2_SHIFT )));
        u32CentreAlignPhCDutyCycle = (uint32)( PWM_DUTY_CYCLE_20KHZ_75
+ ((uint32)((uint32)PWM_DUTY_CYCLE_20KHZ_75 << (uint8)ePWM_DUTY_CYCLE_
MASK2_SHIFT )));
        PwmGeneration( ePWM_CONTROL_REGISTER_ALIGNMENT_CENTRE_ALIGNED,
PWM_PERIOD_20KHZ,
u32CentreAlignPhADutyCycle,  u32CentreAlignPhBDutyCycle,
u32CentreAlignPhCDutyCycle);
        // Now observe activity on SA, SB and SC using a scope.
        break;

        case PWM_EDGE_ALIGNED_DUTY_CYCLE_GENERATION:
         PwmGeneration( ePWM_CONTROL_REGISTER_ALIGNMENT_EDGE_
ALIGNED_MASK, PWM_PERIOD_40KHZ, PWM_DUTY_CYCLE_40KHZ_25, PWM_DUTY_
CYCLE_40KHZ_50, PWM_DUTY_CYCLE_40KHZ_75 );
// Now observe activity on SA, SB and SC using a scope.
        break;

        case PWM_FORCE_LOW_SIDES_ON:
                  ForceAllLowSideOn();
        // Now observe activity on SA, SB and SC are pulled to ground
using a scope.
        break;

        case PWM_GENERATE_NRML_TRIG_SIGNALS:
                  PwmNormalTrigger(PWM_PERIOD_20KHZ, PWM_DUTY_
CYCLE_20KHZ_25);
        // Now observe trigger on GPIO1 and GPIO2
        break;
```

```
        case PWM_GENERATE_CALIB_TRIG_SIGNALS:
        default:
                    PwmNormalTrigger(PWM_PERIOD_20KHZ, PWM_DUTY_
CYCLE_20KHZ_25);
        // Now observe trigger on GPIO3, GPIO4 and GPIO5
        break;
    }
    return E_OK;
}
```

## Edge-Aligned Duty Cycle Generation

This example configures the output PWM period to 40 kHz by assigning PERIOD to 2000 in the PWM period register, sets the PWM mode to edge-aligned by configuring PWMMOD bit to 0 in the control register 1, and configures the dead time to 500 ns by assigning DeadTime to 40 in PWM dead time register. It supports programmable duty cycles as shown in Table 2.

*Table 2: Programmable Duty Cycles*

| Phase | Duty Cycle | Register |
|---|---|---|
| A | 25% | PWM_DUTY_CYCLE_40KHZ_25 |
| B | 50% | PWM_DUTY_CYCLE_40KHZ_50 |
| C | 75% | PWM_DUTY_CYCLE_40KHZ_75 |

Operation in edge-aligned mode depends on the values of DutyCycle1 and DutyCycle2 and falls into the following two conditions:

### Condition 1: DutyCycle1 < DutyCycle2

The duty cycle of the PWM_Compx signal is calculated as follows, depending on the Phasex_Inv signal.

When Phasex_Inv = 0:

$$\mathrm{PWM}_{\mathrm{Compx(Duty\ Cycle)}} = \left(\frac{\mathrm{DutyCycle1x}}{\mathrm{PERIOD}} + \frac{\mathrm{PERIOD-DutyCycle2x}}{\mathrm{PERIOD}}\right) \times 100\,\%$$

When Phasex_Inv = 1:

$$\mathrm{PWM}_{\mathrm{Compx(Duty\ Cycle)}} = \left(1 - \left(\frac{\mathrm{DutyCycle1x}}{\mathrm{PERIOD}} + \frac{\mathrm{PERIOD-DutyCycle2x}}{\mathrm{PERIOD}}\right)\right) \times 100\,\%$$

### Condition 2: DutyCycle1 ≥ DutyCycle

The duty cycle of the PWM_Compx signal is calculated as follows, depending on the Phasex_Inv signal.

When Phasex_Inv = 0:

$$\mathrm{PWM}_{\mathrm{Compx(Duty\ Cycle)}} = \left(\frac{\mathrm{DutyCycle1x}}{\mathrm{PERIOD}} + \frac{\mathrm{PERIOD-DutyCycle2x}}{\mathrm{PERIOD}}\right) \times 100\,\%$$

When Phasex_Inv = 1:

$$\mathrm{PWM}_{\mathrm{Compx(Duty\ Cycle)}} = \left(1 - \left(\frac{\mathrm{DutyCycle1x}}{\mathrm{PERIOD}} + \frac{\mathrm{PERIOD-DutyCycle2x}}{\mathrm{PERIOD}}\right)\right) \times 100\,\%$$

Finally, the output is decided by independent PWM generators when the OutputSelection register is set to value 0x3F.

### Important Notes:

- The operations start once the PERIOD is set to a value greater than 15d.

To execute this configuration, ensure that the variable u8Test is set to PWM_EDGE_ALIGNED_DUTY_CYCLE_GENERATION.

For both edge-aligned and center-aligned modes, the software initialization steps will remain the same, as outlined as follows.

```
void PwmGeneration( enum PwmControlRegisterAlignment eAlignMode, uint16
u16Period, uint32 u32PhADutyCycle, uint32 u32PhBDutyCycle, uint32
u32PhCDutyCycle )
{
        // Set the logic level output to 0 for Hx and Lx signals
        PwmGen_SetOutputLogicLevel(0);
        // Set the dead time to 500ns
        PwmGen_SetDeadTime( PWM_DEAD_TIME_500NS );
        // No change in the signal PWM_Compx
        PwmGen_SetControlRegister1Phase( ePWM_CONTROL_REGISTER_PHASE_NONE );
        // Set the mode of operation
        PwmGen_SetControlRegister1Alignment( eAlignMode );
        // Set PWM Duty Cycle on Phase A
        PwmGen_SetPhaseDutyAFast( u32PhADutyCycle );
        // Set PWM Duty Cycle on Phase B
        PwmGen_SetPhaseDutyBFast( u32PhBDutyCycle );
        // Set PWM Duty Cycle on Phase C
        PwmGen_SetPhaseDutyCFast( u32PhCDutyCycle );
        // Set PWM Period
        PwmGen_SetPeriodFast( u16Period );
        // Enable output selection on all three phases
        PwmGen_SetOutputSelection( ePWM_OUTPUT_SELECTION_ENABLE_ALL_
OUTSEL_XX );
}
```

## Force All Low-Sides On

To force all low-sides on, set all LLVLSEL_Lx bits to 1 in the output logic level selection register and set OUTSEL_Hx and OUTSEL_Lx to 0 in the output selection register as shown below, where X represents phase A, B, or C.

```
void ForceAllLowSideOn( void )
{
    // Set the logic level output to 1 for Lx signal
    PwmGen_SetOutputLogicLevel( ePWM_OUTPUT_SELECTION_LA_LB_LC );
    // Set output selection to zero
    PwmGen_SetOutputSelection( ePWM_OUTPUT_SELECTION_NONE );
}
```

To execute this configuration, ensure that the variable u8Test is set to PWM_FORCE_LOW_SIDES_ON.

## Analog-to-Digital Converter Normal Trigger

In this mode of operation, ADCxNTREF defined by ADC normal trigger register is compared to the carrier signal to generate the trigger signal ADCxNT. The trigger signals for phases B and C are captured through GPIO1 and GPIO2 by configuring

GPIO_USRC1 and GPIO_USRC2 registers with FOSEL = 1 and ESRCSEL = 5. The PWM is generated at 20 kHz with a 75% duty cycle using the edge-aligned mode. The ADC normal trigger for phase B is set at 20% of the PWM period, and for phase C is set at 30% of the PWM period. To enable the ADC normal trigger, set ADCXNT_EN bit 4 and 5 of control register 2.

Note: The ADC normal trigger for phase A cannot be tested, because it requires GPIO0, which is connected to the debugger.

To execute this configuration, ensure that the variable u8Test is set to PWM_GENERATE_NRML_TRIG_SIGNALS and configure period and the duty cycle using the variable u16PwmPeriod and u32PwmDutyCycle, for example, u16PwmPeriod = PWM_PERIOD_20KHZ, u32PwmDutyCycle = PWM_DUTY_CYCLE_20KHZ_75.

```
void InitialiseNormalTrigGpio( void )
{
    // Set GPIOx Control Register to push-pull output.
    Gpio_SetPinActive( eGPIO_INSTANCE_ID1 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID2 );
    Gpio_SetMode( eGPIO_INSTANCE_ID1, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID2, eGPIO_MODE_OUTPUT_PUSH_PULL );

    // Set GPIO_USRCx to have FOSEL = 1 and ESRCSEL = 5
    Gpio_SetSource( eGPIO_INSTANCE_ID1, eGPIO_ESRC5 );
    Gpio_SetSource( eGPIO_INSTANCE_ID2, eGPIO_ESRC5 );
}

void PwmNormalTrigger(uint16 u16PwmPeriod, uint32 u32PwmDutyCycle)
{
    // Initialise GPIO
    InitialiseNormalTrigGpio();
    // Start PWM
    PwmGeneration( ePWM_CONTROL_REGISTER_ALIGNMENT_EDGE_ALIGNED_MASK,
u16PwmPeriod, u32PwmDutyCycle, u32PwmDutyCycle, u32PwmDutyCycle );
    // Normal Trigger at 20% PWM period for phase B
    PwmGen_SetAdcTriggerRef(ePWM_SELECT_B, PWM_NRML_TRIG_20KHZ_20);
    // Normal Trigger at 30% PWM period for phase C
    PwmGen_SetAdcTriggerRef(ePWM_SELECT_C, PWM_NRML_TRIG_20KHZ_30);
    // Enable normal trigger
    PwmGen_SetControlRegister2AdcNormalTrig(ePWM_CONTROL_REGISTER_ADC_
NORMAL_TRIG_ABC_ENABLE);
}
```

## Analog-to-Digital Converter Calibration Trigger

In this mode of operation, ADCxCTREF defined by ADC calibration trigger register is compared to carrier signal to generate the trigger signal ADCxCT signal. The trigger signals for phases A, B, and C are captured through GPIO3, GPIO4, and GPIO5 by configuring GPIO_USRC3, GPIO_USRC4, and GPIO_USRC5 registers with FOSEL = 1 and ESRCSEL = 5.

The PWM is generated at 20 kHz with a 75% duty cycle using the edge-aligned mode. The ADC calibration trigger for phase A is set at 40% of the PWM period, for phase B is set at 50% of the PWM period, and for phase C is set at 60% of the PWM period. To enable the ADC calibration trigger, set ADCXCT_EN bits 6 to 8 of control register 2.

To execute this configuration, ensure that the variable u8Test is set to PWM_GENERATE_CALIB_TRIG_SIGNALS and configure period and the duty cycle using the variable u16PwmPeriod and u32PwmDutyCycle, for example, u16PwmPeriod = PWM_PERIOD_20KHZ, u32PwmDutyCycle = PWM_DUTY_CYCLE_20KHZ_75.

```
void InitialiseCalibTrigGpio( void )
{
    // Set GPIOx Control Register to push-pull output.
    Gpio_SetPinActive( eGPIO_INSTANCE_ID3 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID4 );
    Gpio_SetPinActive( eGPIO_INSTANCE_ID5 );
    Gpio_SetMode( eGPIO_INSTANCE_ID3, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID4, eGPIO_MODE_OUTPUT_PUSH_PULL );
    Gpio_SetMode( eGPIO_INSTANCE_ID5, eGPIO_MODE_OUTPUT_PUSH_PULL );

    // Set GPIO_USRCx to have FOSEL = 1 and ESRCSEL = 5
    Gpio_SetSource( eGPIO_INSTANCE_ID3, eGPIO_ESRC5 );
    Gpio_SetSource( eGPIO_INSTANCE_ID4, eGPIO_ESRC5 );
    Gpio_SetSource( eGPIO_INSTANCE_ID5, eGPIO_ESRC5 );
}

void PwmCalibTrigger(uint16 u16PwmPeriod, uint32 u32PwmDutyCycle)
{
    // Initialise GPIO
    InitialiseCalibTrigGpio();
    // Start PWM
    PwmGeneration( ePWM_CONTROL_REGISTER_ALIGNMENT_EDGE_ALIGNED_MASK,
u16PwmPeriod, u32PwmDutyCycle, u32PwmDutyCycle, u32PwmDutyCycle );
    // Calibration Trigger at 40% PWM period for phase A
    PwmGen_SetAdcCalibTriggerRef(ePWM_SELECT_A, PWM_CALIB_
TRIG_20KHZ_40);
    // Calibration Trigger at 50% PWM period for phase B
    PwmGen_SetAdcCalibTriggerRef(ePWM_SELECT_B, PWM_CALIB_
TRIG_20KHZ_50);
    // Calibration Trigger at 60% PWM period for phase C
    PwmGen_SetAdcCalibTriggerRef(ePWM_SELECT_C, PWM_CALIB_
TRIG_20KHZ_60);
    // Enable calibration trigger
    PwmGen_SetControlRegister2AdcCalibTrig(ePWM_CONTROL_REGISTER_ADC_
CALIB_TRIGGER_ABC_ENABLE);
}
```

# OPERATIONAL WAVEFORMS

This section provides a detailed explanation of the operational waveform for the following scenarios.

## Center-Aligned Duty Cycle Generation

The operational waveform shown in Figure 2 illustrates the signals SA, SB, and SC, with PWM period set to 20 kHz in center-aligned mode. The respective duty cycles for the phases are 25% for SA, 50% for SB, and 75% for SC.

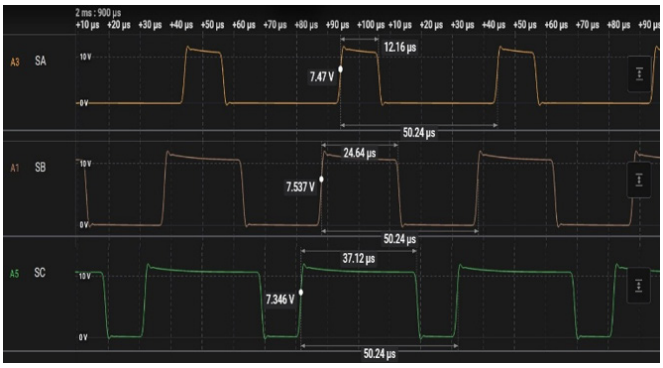


*Figure 2: Center-Aligned Mode Waveform*

## Edge-Aligned Duty Cycle Generation

The operational waveform shown in Figure 3 illustrates the signals SA, SB, and SC, with PWM period set to 40 kHz in the edge-aligned mode. The respective duty cycles for the phases are 25% for SA, 50% for SB, and 75% for SC.

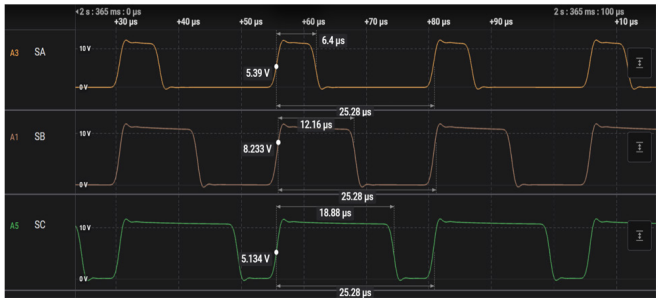


*Figure 3: Edge-Aligned Mode Waveform*

## Force All Low-Sides On

The operational waveform in Figure 4 shows that the signals SA, SB, and SC are pulled low when low-side FET of all three phases (GLA, GLB, and GLC) are turned on.

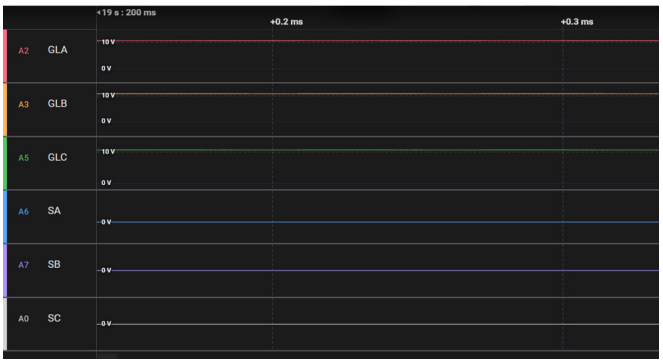


*Figure 4: Low-Side On Waveform*

## Generate ADC Normal Trigger Signal

The operational waveform in Figure 5 shows that the ADC normal trigger captured on GPIO1 for phase B and Figure 6 shows that the ADC normal trigger captured on GPIO2 for phase C, corresponding to 20% of the PWM period (20 kHz) for phase B and 30% for phase C.

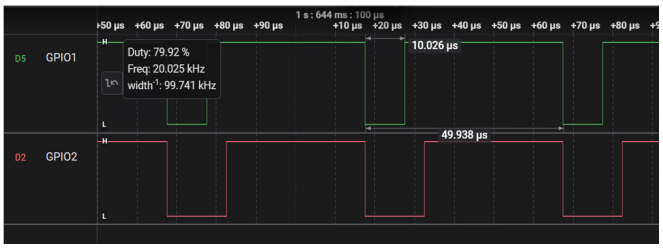


*Figure 5: ADC Normal Trigger on Phase B*

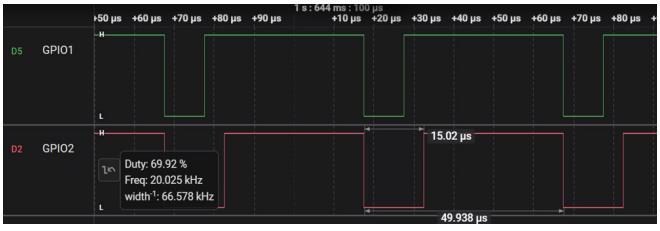


*Figure 6: ADC Normal Trigger on Phase C*

## Generate ADC Calibration Trigger Signal

The operational waveform in Figure 7 shows that the ADC calibration trigger captured on GPIO3 for phase A, Figure 8 captures the trigger on GPIO4 for phase B, and Figure 9 captures the trigger on GPIO5 for phase C, corresponding to 40%, 50%, and 60% of the PWM period (20 kHz) for respective phases.

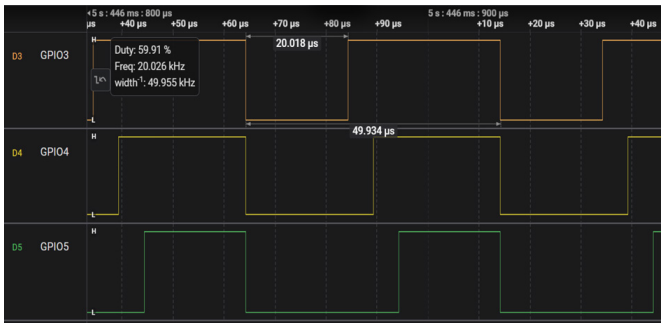


*Figure 7: ADC Calibration Trigger on Phase A*

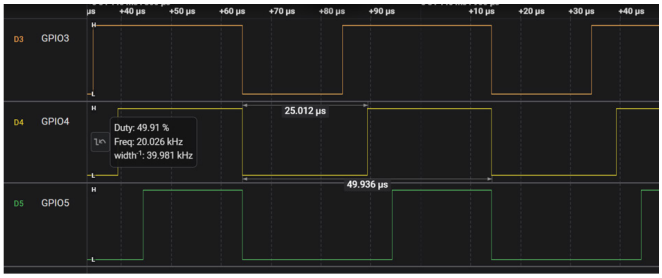


*Figure 8: ADC Calibration Trigger on Phase B*

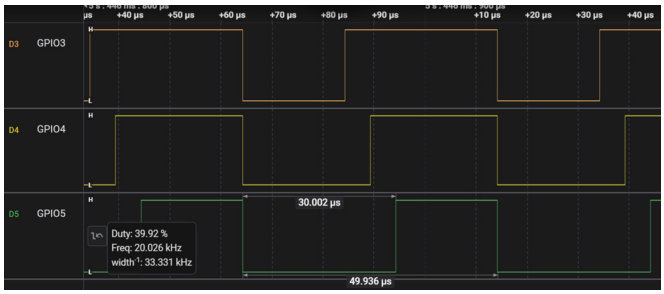


*Figure 9: ADC Calibration Trigger on Phase C*

*Revision History*

| Number | Date | Description |
|---|---|---|
| – | September 15, 2025 | Initial release |
| 1 | November 3, 2025 | Updated PWM example code description (page 1) and added boot capacitor charge section (pages 2-3). |
| 2 | June 29, 2026 | Corrected Center-Aligned Duty Cycle Generation code block (page 4) |



955 PERIMETER ROAD • MANCHESTER, NH 03103 • USA
+1-603-626-2300 • FAX: +1-603-641-5336 • ALLEGROMICRO.COM